Electronic Version
Stylesheet Version v1.1.1

Description

Dense and randomized storage and coding of information

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority from U.S. provisional patent application Ser. No. 60/521,298 filed March 29, 2004.

BACKGROUND OF INVENTION

[0002] The invention relates to data storage and retrieval in electronic forms, particularly to matrix storage and retrieval and the computation of matrix multiplication on electronic computers.

[0003] It is known from the prior art that general n-bit sequences cannot be compressed to a shorter binary sequence or cannot be encoded by less than n bits. However, using some fascinating physical phenomena and different models of computation, superdense coding is still possible. Bennet and Wiesner (C.H. Bennet and S.J. Wiesner. Communication via one- and two particle operators on Einstein-Podolski-Rosen states. Phys. Rev. Lett., 69:2881--2884, 1992.), using Einstein-Podolski-Rosen entangled pairs, showed that n classic bits can be encoded by n/2 quantum bits. However, it is not known how to build machines capable of storing quantum bits.

[0004] Storing and retrieving large amount of data is a challange for today's technology, including computer technology, electronics, medical technology, signal processing, telecommunications technology etc. Because - generally - n bits cannot be compressed to less than n bits, storing twice the amount of data needs at least twice as large storage.

[0005] In many cases data is stored and manipulated in matrices. The storage and the manipulation of matrices is important in any applications involving electronically stored pictures and graphics as well as video. Linear optimization also deals with - generally very large - matrices. One of the most important matrix operations is the matrix multiplication. Matrix multiplcations are computed very frequently on electronic computers. Any improvement on the speed or the storage requirements of the methods of prior art are of the interest of the technical applications.

[0006] References:

[0007] David H. Bailey: Extra high speed matrix multiplication on the Cray-2. SIAM J. Sci. Statist. Comput., 9(3):603--607, 1988.

[0008] C.H. Bennet and S.J. Wiesner. Communication via one- and two particle operators on Einstein-Podolski-Rosen states. Phys. Rev. Lett., 69:2881--2884, 1992.

[0009] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. J. Symbolic Comput., 9(3):251--280, 1990.

[0010] Vince Grolmusz. Computing elementary symmetric polynomials with a sub-polynomial number of multiplications. SIAM Journal on Computing, 32(6):1475--1487, 2003.

[0011] Vince Grolmusz. Near quadratic matrix multiplication modulo composites. Technical Report TR03-001, ECCC, 2003.

[0012] Ran Raz. On the complexity of matrix product. In Proceedings of the thirty-fourth annual ACM symposium on Theory of computing. ACM Press, 2002.

[0013] Volker Strassen. Gaussian elimination is not optimal. Numerische Mathematik, 13:354--356, 1969.

SUMMARY OF INVENTION

[0014] Let m be a non-prime power, composite integer. For example, we may choose m=6. Let $x_1,x_2,\ldots,x_n$ be a sequence of n variables. Then, using linear transforms modulo 6, we compute $t=n^{o(1)} << n$ linear forms of the x_i variables, denoted by $z_1,z_2,\ldots,z_t$, such that using another linear transform to these z_j's, we get back the a representations of the x_i's, namely $x_i+3(x_{i1}+x_{i2}+\ldots+x_{i}\_u)+4(x_{j1}+x_{j2}+\ldots+x_{jv})$, for i=1,2,...,n, where the different indices mean different numbers (Quantity o(1) here denotes a positive number which goes to 0 as n goes to infinity). From our representation one can get back the actual x_i's by using filtering, described also in this disclosure.

[0015] We apply the method, illustrated above to dense coding and storage of information, dense coding and storage of numbers and matrices, and also we apply the method to a very fast computation of a representation of the matrix product.

Brief Description of Drawings

[0016] FIG. 1 is a flow chart illustrating the dense coding, storage and decoding of information.

[0017] FIG. 2 is a flow chart illustrating the dense coding, storage and decoding of matrices.

[0018] FIG. 3 is a flow chart illustrating the matrix multiplication method of the present disclosure.

[0019] FIG. 4 is a flow chart illustrating an alternative method for dense coding, storage and decoding of matrices.

[0020] FIG. 5 gives examples for matrices A, B and C.

[0021] FIG. 6 gives examples for step-functions and the linear combination of step-functions.

DETAILED DESCRIPTION

[0022] The present invention is described in part in Vince Grolmusz: Defying Dimensions Modulo 6, Electronic Colloquium on Computational Complexity Report TR03- 058, (ISSN 1433-8092), Revision 2 (2004) which is incorporated herein by reference.

[0023] A general method was shown in (Vince Grolmusz: Low Rank Co-Diagonal Matrices and Ramsey Graphs, Electronic Journal of Combinatorics, Vol. 7, (2000), No. 1, R15) for the construction of n x n matrices A' with 0's in the diagonal and non-zeroes elsewhere modulo a non-prime power integer, denoted by m. Said construction has the main property that said matrices have small rank modulo m, that is, matrix A' can be written as the matrix product B'C' modulo m, where B'is an n x (t-1) and C' is a (t-1) x n matrix with integer elements, where t is a small number relative to n, that is, $t=n^{o(1)}$, where o(1) denotes a positive quantity which goes to 0 as n goes to the infinity.

[0024] It is also known from the prior art, that said matrix A' can be constructed that way, that if m has distinct prime divisors p_1, p_2,...,p_r, then the non-zero elements of matrix A' are either 0 or 1 modulo p_i, for i=1,2,...,r. For example, if m=6, then the non-zero elements of matrix A' are either 3 or 4, modulo 6.

[0025] Let J denote the n x n all-1 matrix. Let us consider the matrix A=J-A'. It contains 1's in the diagonal, and numbers, congruent to zero modulo at least one prime divisor of m. Returning to the previous example, with m=6, we have that A has either 3 or 4 or 0 outside of the diagonal. Matrix A can be written as the matrix product BC modulo m, where B is an n x t and C is a t x n matrix with integer elements.

[0026] FIG. 6 gives an example with m=6, n=16, t=13 for matrices A, B and C.

[0027] There are several other ways to construct matrices with similarly useful properties than that of A. Such method is known from the prior art (e.g.,Vince Grolmusz: A Note on Explicit Ramsey Graphs and Modular Sieves, Combinatorics, Probability and Computing Vol. 12, (2003) pp. 565-569). Another way is to construct matrix A is as follows: the entry in row i and column j of matrix A is defined as the Hamming-distance of the binary forms of numbers i and j. By this definition we get matrices B and C such that A=BC, where B is an n x t and C is a t x n matrix with integer elements, and t=O(log n).

[0028] The larger the quantity n is, the smaller the quantity t becomes, relative to n. This property is used in the present disclosure to compactify electronically stored information.

[0029] Let $x=(x_1,x_2,\ldots,x_n)$ be a sequence of n variables. We compute $t=n^{o\ (1)}$ linear forms of the x_i's, denoted by $z=(z_1,z_2,\ldots,z_t)$, such that using another linear transform to this z, we get back a representation of the x. More exactly, Let A=BC. Then let z=xB, and x'=zC=xBC=xA.

[0030] FIG. 1 llustrates our method for compacting information. Compacting process 101 involves computing z=xB modulo m. The t coordinates of z can be stored (102) in a much smaller place than x. The expanding process 103 is computing zC, also modulo m. The filtering process 104 is meant to recover the original coordinates of the vector x from the vector x'=zC=xBC=xA.

[0031] FIG. 2 illustrates our method for compacting matrices. Let X be an n x n matrix. The row compacting process 201 is done by computing the n x t matrix XB. The column compacting process 202 is done by computing B^TXB, that is, a t x t matrix. In the storage 203 step this small matrix is stored. Retrieving the matrix is done in three phases: 204, 205, 206. The column expanding process 204 is done by computing C^TB^TXB, and the row expanding process 205 is done by computing the n x n matrix $X'=C^TB^TXBC$. The filtering process 206 is meant to recover the original matrix X from X'.

[0032] FIG. 4 illustrates an alternative method for compacting matrices. Let X be an n x n matrix. The column compacting process 401 is done by computing the t x n matrix B^TX. The row compacting process 402 is done by computing B^TXB, that is, a t x t matrix. In the storage 403 step this small matrix is stored. Retrieving the matrix is done in three phases: 404, 405, 406. The row expanding process 404 is done by computing B^TXBC, and the column expanding process 405 is done by computing the n x n matrix $X'=C^TB^TXBC$. The filtering process 406 is meant to recover the original matrix X from X'.

[0033] We disclose three methods for performing the filtering process (items 104, 406, 206). All of these methods aim to cancel out those variables or entries from the retrieved numbers or matrices, which have coefficients other than 1. This is necessary, since we get back xA=xBC in the case of the compacting-expanding process of FIG. 1, and A^TXA in the case of matrix compacting and expanding in FIG. 2 and FIG. 4.

[0034] The first method for filtering is applicable for 0-1 values of variables $x=(x_1,x_2,...,x_n)$. Here we change the values of the coordinates with value 1 to 0, one by one, and observe the change of the values of the variable x'_i, for i=1,2,...,n. Any change which is not -1 is neglected, and a change in x'_i of value -1 means that the original value of x_i was 1.

[0035] The second method involves a device called filter-machine. First, let m=6, and G(x)=f(x)+3g(x)+4h(x) be a polynomial of n variables $x=(x_1,x_2,...,x_n)$, where the coefficient of every monomial in f is 1 modulo 6, and no monomial appears in two of the polynomials f, g, h with non-zero coefficients modulo 6. Then M is a mod 6 filter-machine for polynomial G(x), if for inputs polynomial G(x) and number G(s) for some s element of {0,1,2,3,4,5}<superscript>^{n}</superscript>, M returns the value f(s) mod 6. With more details, the filter machine works as follows. Let us consider polynomial G(x), and suppose that we can repeatedly increment the value of polynomials f, g, and h, independently by 1. Then the period of 3g is 2, the period of 4h is 3, while the period of f is 6, all seen modulo 6. So if we were able to filter out the shorter period (that is, the higher frequency) waves then we were able to compute f(s). Note, that for doing this we need to substitute values from Z_6 instead just bits into the polynomials. Note also, that machine M does not need access to the actual values of the variables of the polynomials g and h, it just needs access to their periodically changed values. After filtering g and h out, the machine asks for the value of G(s), reduced by this filtering, which is just f(s). Let us see the most important example: the dot-product. Suppose that we would like to retrieve the value of x_1= s_1 from set {0,1}. As the first coordinate of zC, we get x'_1=x_1+3(x_{i1}+x_{i2}+...+$x_{i\,s}$)+4(x_{j1}+x_{j2}+...+$x_{j\,k}$), where i_u is not equal to j_v, u=1,2,...,s, v=1,2,...,k. Now, M is assumed to have access to some values of x_{i1}+x_{i2}+...+x_{i} $_{s}$ in order to filter them out, since their periodicity is at most 2 modulo 6; and also to some values of x $_{j1}$</subscript>+x_{j2}</subscript>+...+x_{j} $_{k}$</subscript> to filter them out, since their periodicity is at most 3 modulo 6. Note again, that their s_i</subscript> values are not needed at this phase, and, also, that typically, it is not enough to substitute 0 and 1 in the variables, 0,1,2,3,4,5 may be needed. After identifying the higher frequency terms, M filters them out, and returns the value of f, which is s_1 in our case. Note, that we ask only here for the value of a variable. In the general case, when m differs from 6, but a non-prime power composite, we have seen that the non-diagonal elements of matrix A are zero for at least one prime divisor of m. That means that the period of those variables which have this non-diagonal element as coefficient, is a divisor of m, so in general an analogous method works for filtering out the variables with smaller periodicity coefficients.

[0036] The third filtering method disclosed involves randomized memory cells. Let h_1 ,h_2,...,h_n be the n bits to be stored. Then let s_1,s_2,...,s_n be step-functions (i.e., piece-wise constant functions) on the [0,1] interval, such that the following properties hold:

[0037] Function s_i has finitely many, but at least one non-zero steps modulo m, for i=1,2,...,n. In addition, a step of function s_i is either 0 modulo m or it is non-zero modulo all individual prime-divisors of number m, for i=1,2,...,n,

[0038] No two different functions s_i and s_k have non-zero steps modulo m in the same point r in the real interval [0,1].

[0039] Note, that the last requirement can easily be met if the step-functions are chosen randomly.

[0040] FIG. 6 gives examples of step-functions S (item 601) and Z (item 602). The step of S is -1 at point 604, the step of Z is also -1 at step-point 605, while 2S+3Z (item 603) has step in 605A of value -2 and step in point 604A of value -3.

[0041] Now we describe the randomized memory, capable for high-density data storage. Bit h_i is corresponded to $x_i=h_is_i$, for i =1,2,...,n. Then we apply matrix B to x by computing z=xB. Note, that the linear combination of the step-functions modulo m are also step functions. Then the randomized storage of the bits $h_1,h_2,\ldots,h_n$ is done by the randomized piecewise constant functions $z_1,z_2,\ldots,z_t$. From the z_i's, we compute x'_i's as x'=zC=xBC. We then observe the change of the values of the piecewise constant function x_i'. If every change (that is, steps) of the function x_i' is 0 modulo at least one prime divisor of m, then $h_i=0$, otherwise, $h_i=1$. The advantages of the method: One need not know the s_i, so s_i can be some physical object (elementary particle state), one should only apply linear transforms to it, store it, then apply another linear transform to it, and observe only the the change of the function (i.e., the spectrum of the emitted photons by the particle).

[0042] FIG. 3 describes the fast matrix multiplication method. A method is disclosed for computing the representation of the product of the n x n matrix X and the n x n matrix Y with only $n^{o(1)}$ multiplications. The column compacting 302 of matrix X is done by computing $B^T X$. The row compacting 301 of Y is done by computing YB. Next, a certain product 303 of the t x n matrix $B^T X=\{u_{ij}\}$and the n x t matrix $YB=\{v_{kl}\}$ is computed as follows. The t x t matrix $W=\{w_{il}\}$ is computed as

[0043] $w_{il}=\Sigma^{t}_{j=1} (\Sigma^{n}_{k=1} b_{kj} u_{ik}) (\Sigma^{n}_{k=1} c_{jk} v_{kl})$.

[0044] Computing w_{il} uses only t non-scalar multiplications, consequently, computing matrix W uses only $t^3 = n^{o(1)}$ multiplications.

[0045] Column expanding process 304 is done by computing $C^T W$, and the row expanding process 305 is done by computing $C^T WC$. Steps 304 and 305 can be exchanged in the method. At the end a filtering process 306 is done for retrieving the values of the product matrix.

[0046] Although the subject invention has been described with respect to particular embodiments, it will be readily apparent to those having ordinary skill in the art to which it pertains that changes and modifications may be made thereto without departing from the spirit or scope of the subject invention as defined by the appended claims.